Software Tool for Random-Based Generation of Switching Function Benchmarks

Miloš M. Radmanović[1]

***Abstract* – Traditional approach to the measurement of performance for CAD algorithms involve the use of sets of so-called "benchmark functions". This paper describes a system (software tool), that generates and manipulates random constraint two-level representation of a Boolean function corresponding to the so called "real-world" functions. The software tool satisfies both the requirements to build some common benchmarks useful to compare different research results, and to create a tool for supporting intensive test of new algorithms. The paper gives an overview of the functionalities of the software tool and describes parameters that characterize the layout of a function. Tool is made publicity available in an attempt to extend standard sets of benchmark functions.**

***Keywords* – Synthetic random benchmark generator, switching functions, Berkeley PLA format, software tool.**

I. INTRODUCTION

Benchmark functions design is the basis for the performance evaluation of today's CAD algorithms. The most commonly method to verify the competitiveness of a new algorithm consists of applying this algorithm to a set of benchmark functions design in a given experimental settings. The experimental results are then compared to those obtained by applying a comparable algorithm to the same set of benchmarks design. This implies that the quality of CAD algorithm's evaluation is only as good as the functions design is used for benchmarking.

In recent years, there have been several initiatives for assembling benchmark design suits consisting of two-level representation of Boolean functions to improve the quality of CAD algorithms. An extensive research on benchmarks was conducted in both academia and industry. Industry use real customer designs as benchmarks to demonstrate performance of their products over their competition. However, customer designs are usually confidential and provided under non-disclosure agreements. Most of the benchmarks in academia originate from conferences and workshops. For instance MCNC.91 [5] and IWLS.05 [6] were published for workshops on Logic Synthesis. The MCNC.91 benchmark suite has standardized libraries with representative circuit designs ranging from simple circuits to advanced circuits obtained from industry. MCNC benchmarks are very popular in academic research. The IWLS.05 contains diverse circuit designs derived from past conference benchmarks, open source community of hardware designers, and industry to represent a variety of applications. The benchmarks were synthesized and organized into a standardized library with a common timing infrastructure, standard interfaces and reporting formats to promote easy exchange of benchmarks and experimental results in the community. Conference benchmarks are widely circulated because they are freely available in public domains. Open source communities allow members to share benchmarks, methodologies, and results. For instance OpenCores [7] is an open source community that deals with semiconductor intellectual property cores. Academia and industrial corporations use freely available OpenCores designs to benchmark their products.

In some cases, benchmark designers turn to automatic generation of synthetic benchmarks to evaluate new architectures which they can't efficiently test using existing benchmarks [10]. The paper [8] presents an approach to generate synthetic benchmarks for evaluating new architectures and tools, which don't have representative evaluation benchmark sets. For instance, the paper [9] implemented an algorithm for generating synthetic benchmarks and used them to study optimality and scalability of placer tools.

The problem of a correct experimental evaluation is crucial in several areas of Logic synthesis, in particular when a computationally intensive test of specific algorithms is needed. This paper describes a software tool (synthetic benchmark generator), that generates and manipulates random constraint two-level representation of a Boolean function corresponding to the so called "real-world" functions. My attention to the problem generates from previous work on the evaluation of the specialized algorithms for calculation over decision diagrams [1], [2]. This previous work has shown that evaluation of some algorithms needs for reliable benchmarks, like a library of similar problems having different characteristics. In particular, I address the problem of defining a set of random-based two-level representation of a Boolean function according to a number of well designed parameters and their management by a software tool. Software tool generates files in widely used Berkeley PLA (Espresso) format [13] for two-level representation of Boolean function. Tool is made publicity available [15] in an attempt to extend standardized libraries of benchmark functions.

There are many different approaches to generate random-based benchmarks [3], [4], [8], [11], [12]. When in need of random-based two-level representation of a Boolean function benchmarks some approach can be followed. The first approach consists of random generation of two-level representation of a Boolean function with the probability of appearance for each element of the representation. The second approach consists of random appearance of the basic logic

[1]Miloš M. Radmanović is with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: milos.radmanovic@gmail.com

operation sequences and the third approach consists of random appearance of the arithmetic logic unit operation sequences.

This paper is organized as follows: Section 2 shortly introduces the Berkeley PLA (Espresso) format for two-level representation of Boolean function; Section 3 presents the technical details of the random-based benchmarks generator proposed in this paper; Section 4 describes the software tool capabilities to offer an experimental workbench and gives some examples of using generated benchmarks. Some concluding remarks end the paper.

II. BERKLEY PLA FORMAT

The Berkley PLA (Espresso) format is a logical representation of a set of Boolean equation. The format has been expanded to allow for multiple-valued logic functions, and to allow for the specification of the don't-care set. Programs exist to translate a set of equations into this format (e.g., eqntott, bdsyn, eqntopla). PLA format is described as a character matrix with keywords embedded in the file to specify the size of the matrix and the logical format of the function.

The minimum required set of keywords is: *.i* (specifies the number of input variables) and *.o* (specifies the number of output variables) for binary-valued functions, or *.mv* for multiple-valued functions. A complete list of the keywords is given in [13].

The default PLA file formats are compatible with the Berkeley standard format for the physical description of a PLA (Programmable logic array). It is generally assumed that the PLA format is specified such that each row of the PLA fits on a single line in the file. A term is represented by a "cube" which can be considered either a compact representation of an algebraic product term which implies the function value is a 1, or as a representation of a row in a PLA which implements the term. A cube has an input part which corresponds to the input plane of a PLA, and an output part which corresponds to the output plane of a PLA. Each position in the input plane corresponds to an input variable where a *'0'* implies the corresponding input literal appears complemented in the product term, a *'1'* implies the input literal appears uncomplemented in the product term, and '-' implies the input literal does not appear in the product term. For each output, a literal *'1'* means that this product term belongs to the ON-set, a *'0'* means that this product term belongs to the OFF-set, a '−' means that this product term belongs to the DC-set, and a '~' implies this product term has no meaning for the value of this function. The ON-set of a Boolean function is defined as the set of minterms for which function value is a 1. The OFF-set of a Boolean function is defined as the set of minterms for which function value is a 0. The DC-set (don't care set) is defined as the set of minterms for which the function value is unspecified. A function is completely described by providing its ON-set, OFF-set and DC-set. A complete description of PLA format is given in [13].

The Berkley PLA format representation is illustrated in Figure 1 by a two-input adder.

```
.i 4
.o 3
.p 11
100- 010
0101 010
001- 010
1-00 010
0-10 010
1111 010
-111 100
11-1 100
-1-0 001
-0-1 001
1-1- 100
.e
```

Fig. 1. Example of Berkley PLA format

III. SOFTWARE TOOL

In this section I described the basic ideas followed to generate random two-level representation of Boolean function based on Berkley PLA format. The procedures build two-level representation by using a set of parameters to control the structure of the cubes.

The first generation algorithms controls nine two-level representation's characteristics: (1) number of input variables, (2) number of output variables, (3) number of cubes, (4) probability of appearance for input literal '0', (5) probability of appearance for input literal '1', (6) probability of appearance for input literal '-', (7) probability of appearance for output literal '0', (5) probability of appearance for output literal '1', (6) probability of appearance for output literal '~'. I believe that changing these characteristics it is possible to generate a significantly representative subset of similar two-level representation having different characteristics. The proposed generation algorithm is able to generate either a large benchmark sets with several levels of difficulty to test average performance of a reasoning algorithm or to create a particular instance to test such an algorithm in extreme situations.

The basic idea is to randomly map set of input literal '0' on set of input part of cube. The random mapping is controlled by literal '0' density (probability of appearance). The other aspects to control random mapping are input literal '1', input literal '-', output literal '0', output literal '1' and output literal '~' density.

The actual generating algorithm can produce only synthetic benchmark functions not corresponding to the so called "real-world" functions. I am now modifying it for producing also benchmark functions corresponding to the "real-world" functions.

The second and the third generation algorithms controls six two-level representation's characteristics: (1) number of input variables, (2) number of output variables, (3) number of cubes, (4) probability of appearance for input literal '0', (5) probability of appearance for input literal '1', (6) layout of output ON-set and OFF-set.

The second generation algorithm proceeds similarly to the first generation algorithm with two notable differences. First, I don't use random mapping of input literal '-'. Second, ON-set and OFF-set of a Boolean function is defined as a basic logic

operation (AND, OR, NOT, NAND, NOR, EXOR and EXNOR) sequence.

In order to guarantee the "real-world" benchmark functions, the third generation algorithm extends the second generation algorithms by defining ON-set and OFF-set as a arithmetic-logical operation (adder, multiplier) sequence.

To exploit the capabilities of generating random two-level representation of Boolean function, I have inserted it in a software tool (synthetic benchmark generator) that allows a flexible interaction between a user and generation algorithms.

Software tool is written in MS Visual C++ and use MFC technology [14]. It consists of two basic modules: (1) Random two-level representation generator and (2) Interaction module that allow user to interact with two-level representation in Berkley PLA format and to control representation's characteristics as previously described. It is to be noted constant possibility of naming and saving a current generated function, of choosing particular generating algorithm to be used and of additional editing a two-level representation in PLA format.

IV. Using the Generated Benchmarks

Binary Decision Diagrams (BDDs) are data structures convenient for representation of discrete functions. BDDs are derived by the reduction of the corresponding binary decision trees (BDTs). The reduction is performed by sharing the isomorphic subtrees and deleting the redundant information in the BDT using the suitably defined reduction rules. BDDs are often substantially more compact than traditional normal forms such as conjunctive normal form and disjunctive normal form. They can also be manipulated very efficiently. Hence, BDDs have become widely used for a variety of CAD applications, including symbolic simulation, verification of combinational logic and verification of sequential circuits.

Multiple-output switching functions are represented by shared BDDs (SBDDs) [16] having a separate root node for each output. Thus, SBDDs are obtained by sharing isomorphic subtrees in BDDs for outputs of function, considered as separate particular switching functions.

I now show examples of ability of creating subset of similar two-level representation having different characteristics. Below I give tables of different time and space SBDD testing statistics with similar generated two-level representations of Boolean functions. I performed the testing on a PC Pentium IV on 2,66 GHz with 4 GB of RAM (MS Windows 7 Ultimate). The memory usage for all tests was limited to 2 GB.

It is now possible to show the effectiveness of the work perform by the generation algorithms.

Table 1 describes SBDD time and space statistics using generated benchmark based on the first algorithm controlled by changing the number of inputs and the density of input literal '0'. In most cases it is shown that low density of input literal '0' in two level representation of Boolean function produce more SBDD nodes. Differences in performances between functions with different number of inputs can be evaluated from generated benchmarks. Several classes of functions can be compared under control of some parameters.

Table 2 describes SBDD time and space statistics using generated benchmark based on the first algorithm controlled by changing the number of inputs and the density of output literal '1'. In most cases it is shown that medium density of output literal '1' in two level representation of Boolean function produce more SBDD nodes. Also, it is shown that extremely low and high density of literal '1' increase the size of SBDD.

Table 3 describes SBDD time and space statistics using generated benchmark based on the third algorithm controlled by changing number of inputs, outputs and cubes. In most cases it is shown that large number of cubes produces more SBDD nodes.

I encountered a number of factors that requires repetitions of experiments. The major factor is non trivial, unexpected and often unexplainable variability of results under slightly different parameters condition.

I just report the fact that first experimentation I am performing is quite satisfactory. It opens the possibility to create a number of random benchmarks with similar parameters and all different on random basis.

TABLE I
SBDD TIME AND SPACE STATISTIC USING GENERATED BENCHMARKS BASED ON FIRST ALGORITHM (DENSITY OF INPUT LITERAL '0')

Function name	inputs/outputs/cubes /density of inp. literal '0'	time [s]	nodes
rnd0_01	75/100/150/5	53.24	1635332
rnd0_02	75/100/150/15	2.94	183136
rnd0_03	75/100/150/25	1.11	71960
rnd0_04	75/100/150/35	0.68	39766
rnd0_05	75/100/150/45	0.54	37683
rnd0_06	75/100/150/55	0.5	32185
rnd0_07	75/100/150/65	0.42	26346
rnd0_08	75/100/150/75	0.4	26755
rnd0_09	75/100/150/85	0.53	28517
rnd0_10	75/100/150/95	0.61	24397
rnd0_11	100/100/150/5	104.02	2352778
rnd0_12	100/100/150/15	3.66	214117
rnd0_13	100/100/150/25	1.24	73326
rnd0_14	100/100/150/35	0.74	46874
rnd0_15	100/100/150/45	0.48	33324
rnd0_16	100/100/150/55	0.46	30385
rnd0_17	100/100/150/65	0.48	32200
rnd0_18	100/100/150/75	0.46	32505
rnd0_19	100/100/150/85	0.56	31984
rnd0_20	100/100/150/95	0.98	38543
rnd0_21	150/100/150/5	54.05	1785260
rnd0_22	150/100/150/15	3.08	181595
rnd0_23	150/100/150/25	1.32	76929
rnd0_24	150/100/150/35	0.67	44976
rnd0_25	150/100/150/45	0.65	43895
rnd0_26	150/100/150/55	0.49	37768
rnd0_27	150/100/150/65	0.64	41911
rnd0_28	150/100/150/75	0.54	38098
rnd0_30	150/100/150/85	0.62	42794
rnd0_31	150/100/150/95	1.35	62956

TABLE II
SBDD TIME AND SPACE STATISTIC USING GENERATED BENCHMARKS BASED ON FIRST ALGORITHM (DENSITY OF OUTPUT LITERAL "1")

Function name	inputs/outputs/cubes /density of inp. literal '0'	time [s]	nodes
rnd1_01	75/100/150/5	0.09	11282
rnd1_02	75/100/150/15	0.25	22808
rnd1_03	75/100/150/25	0.53	39326
rnd1_04	75/100/150/35	0.89	49733
rnd1_05	75/100/150/45	0.96	56972
rnd1_06	75/100/150/55	1.34	57661
rnd1_07	75/100/150/65	0.95	43908
rnd1_08	75/100/150/75	1.51	59644
rnd1_09	75/100/150/85	1.17	50242
rnd1_10	75/100/150/95	0.62	28826
rnd1_11	100/100/150/5	0.11	13891
rnd1_12	100/100/150/15	0.28	26655
rnd1_13	100/100/150/25	0.53	41679
rnd1_14	100/100/150/35	0.68	44775
rnd1_15	100/100/150/45	1.03	57787
rnd1_16	100/100/150/55	0.91	51122
rnd1_17	100/100/150/65	1.26	57370
rnd1_18	100/100/150/75	1.41	59082
rnd1_19	100/100/150/85	0.96	40281
rnd1_20	100/100/150/95	0.48	23023
rnd1_21	150/100/150/5	0.14	18338
rnd1_22	150/100/150/15	0.31	32156
rnd1_23	150/100/150/25	0.51	44581
rnd1_24	150/100/150/35	0.84	58650
rnd1_25	150/100/150/45	0.84	56385
rnd1_26	150/100/150/55	1.21	67633
rnd1_27	150/100/150/65	1.32	62448
rnd1_28	150/100/150/75	1.23	55816
rnd1_30	150/100/150/85	1.38	57329
rnd1_31	150/100/150/95	0.62	34542

TABLE III
SBDD TIME AND SPACE STATISTIC USING GENERATED BENCHMARKS BASED ON THIRD ALGORITHM

Function name	inputs/outputs/cubes	time [s]	nodes
rnd_mul32_01	64/64/250	0.29	17675
rnd_mul32_02	64/64/500	0.81	34727
rnd_mul32_03	64/64/750	1.59	50998
rnd_mul32_04	64/64/1000	2.74	67391
rnd_mul48_01	96/96/250	0.43	27642
rnd_mul48_02	96/96/500	1.27	54327
rnd_mul48_03	96/96/750	2.59	80574
rnd_mul48_04	96/96/1000	4.38	106761
rnd_mul60_01	120/120/250	0.57	35098
rnd_mul60_02	120/120/500	1.63	69059
rnd_mul60_03	120/120/750	3.35	102815
rnd_mul60_04	120/120/1000	5.85	135944

.

V. CONCLUSION

This paper describes a software tool that generates and manipulates random constraint two-level representation of a Boolean function in Berkley PLA format corresponding to "real-world" functions. The tool includes three approaches to build random two-level representation by using a set of parameters to control the structure of the cubes.

The tool satisfies both the requirements to build some common benchmarks useful to compare different research results, and to create a tool for supporting intensive test of new algorithms.

REFERENCES

[1] M. Radmanović, "Tools for Calculating Autocorrelation Spectrum by Using The Wiener-Khinchin Theorem", 39th Int. Conf . ICEST 2006, Sofia, 2006.

[2] M. Radmanović, R. Stanković, C. Moraga, "Analysis of Decision Diagram based Methods for the Calculation of the Dyadic Autocorrelation", International Journal of Systemics, Cybernetics and Informatics, Pentagram Research Publication, July 2007.

[3] S. Bhawmik, V.K. Narang and P. Pal Chaudhuri, "Selecting Test Methodologies for PLAs and Random Logic Modules in VLSI Circuits - An Expert System Approach", The VLSI Journal, Volume 7, Issue 3, Pages 267-281, September 1989.

[4] K. Iwama, S. Sawada, K. Hino, H. Kurokawa, "Random Benchmark Circuits with Controlled Attributes", In. Proc. of European Design and Test Conference 97, 90 – 97, 1997.

[5] F. Brglez, "ACM/SIGDA Benchmarks Electronic Newsletter", DAC 93 Edition. June 1993, 1-22, http://www.cbl.ncsu.edu/benchmarks.

[6] C. Albrecht, "IWLS 2005 Benchmarks," 2005. http://iwls.org/iwls2005/benchmark_presentation.pdf

[7] "OpenCores", www.opencores.org/.

[8] P. Verplaetse, J. Campenhout, and D. Stroobandt, "On Synthetic Bencmark Generation Methods," In. Proc. of IEEE In. Symposium on Circuits and Systems, vol. 4, pp. 213-6, 2000.

[9] C. Chang, J. Cong, M. Romesis, and M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms," IEEE Transactions on CAD of Integrated Circuits and Systems, vol. 23, no. 4, pp. 537-49, 2004.

[10] R Njuguna, "A Survey of FPGA Benchmarks", Project Report, http://www.cs.wustl.edu/~jain/cse567-08/ftp/fpga/

[11] J. Darnauer and W. Dai, "A Method for Generating Random Circuits and Its Application to Routability Measurement", In. Proc. of the 1996 ACM 4. In. Symposium on FPGA, 66 – 72, 1996.

[12] J. Harlow and F. Brglez, "Design of Experiments for Evaluation of BDD Packages Using Controlled Circuit Mutations", In Proc. of the In. Conference on Formal Methods in CAD, 64-81, 1998.

[13] R. Rudell, "Espresso Misc. Reference Manual Pages", University of California, Berkeley, California, USA, http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm

[14] Microsof Visual Studio 6.0, Microsoft Corporation, http://msdn.microsoft.com/en-us/library/ms950417.aspx

[15] Stgroup CIITLAB Software http://stgroup.elfak.edu.rs/home/node/4

[16] T. Sasao T, M. Fujita M, "Representations of discrete functions. Boston", Kluwer Academic Publishers, 1996.